IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-7, Issue-10, Oct 2019 E-ISSN: 2347-2693

Enhanced K_way Method In "APRIORI" Algorithm for Mining the Association Rules Through Embedding SQL Commands

Basel A. Dabwan[1*], Mukti E. Jadhav[2]

[1]Dept. of Computer Science, Dr. Babasaheb Ambedkar Marathwada University, Aurangabad, India
[2]Dept. of Computer Science, M.I.T.College, Aurangabad, India

[*]*Corresponding Author: baseldbwan@yahoo.com, Tel.: +91-8828196252*

DOI: https://doi.org/10.26438/ijcse/v7i10.5256 | Available online at: www.ijcseonline.org

Accepted: 14/Oct/2019, Published: 31/Oct/2019

Abstract— No doubt, the notable and bursting growth in data and databases has produced an imperative necessity for new mechanism and devices that can rationally and spontaneously convert the handled data into helpful and valid information and knowledge. Data mining is such a style that evolves non axiomatic, tacit, formerly anonymous, and possibly beneficiary information from data in databases. In this paper we achieved some Enhancements in K_way Method In "APRIORI" Algorithm for Mining the Association Rules Through Embedding SQL Commands.

Keywords— Ddata mining; association rules; relational, database; Apriori ; SQL.

I. INTRODUCTION

The quick growth in data and databases has created a pressing need for new tools and techniques that can quickly and efficiently process raw data into useable information . Mining association is one of the regularly used methods in data mining. Basket data analysis is typical of this method.
There are some approaches proposed to mine association rules in relational databases using SQL [1,2,3,4,5,6,7,8,9,10]. Different efforts have been proposed to enhance database-related association rule mining. There have been proposedlanguage extensions of SQL to support mining operations; for instance, in [7], DMQL extends SQL with a series of operators forgeneration of characteristic rules, discriminant rules and classification rules. In [4], an inductive method using SQL has been presented to generate frequent itemsets. In [5], the cost of generating association rule for a given item in object relational databases has been tested to show that the cost of computing association rules does not depend on support and confidence threshold. In [6], an improved algorithm for mining association rules based on computing Sequence Number Degree (SND) has been introduced [10] presents the Btree index, as an approach to provide tight integration of item set extraction in a relational DBMS.

Since the generation of frequent itemsets is the most expensive part of association rule mining, in this work we introduce an enhanced method for frequent item set generation by rewriting the SQL statement of frequent item set generation in such a way that avoid joining the item to itself as this step is not logic and consumes more time and resources. Our approach shows significant performance enhancement in terms of query execution time which is less than traditional K-way method.

This paper is organized as the following: an introduction and related work are presented in Section I. the definition of Association rules is presented in Section II. Apriori Algorithm is presented in Section III. In section IV, our enhanced method on K-way approach with related experiments and results are presented. Finally, conclusions are presented in section V.

II. DEFINITION OF ASSOCIATION RULES

Relationships between unrelated data in a targeted data sources can be stated as if/then statement. An example of an association rule would be "If someone buys an apple, he is 75% likely to also purchase orange. "

The formal definition of Association rules mining provided by [1] is as follow:
Let **I** = {iB_{1B}, iB_{2B},.., iB_{mB}} be a set of literal, called **items**. Let **D** = {tB_{1B}, tB_{2B},.., tB_{nB}} be a set of transactions, where each transaction, t, is a set of items such that $t \subseteq$ I. Note that the quantities of items in a transaction are not considered. Each transaction is associated with an identifier, called TID. Given an itemset $X \subseteq$ I, a transaction t **contains** X if, and only if, $X \subseteq t$.

The itemset X has support, **s,** in the transaction set D if s% of transactions in D contain X; we denote s = support (X). An association rule is an implication of the form X => Y,

where X, Y ⊂ I, and X ∩ Y = Φ.Each rule has two measures of value, support, and confidence. The **support** of the rule X => Y is support (X ∪ Y). The **confidence, c,** of the rule X => Y in the transaction set D means c% of transactions in D that contain X also contain Y, which can be written as the ratio support (X ∪ Y) / support (X). this confidence gives an indication how strong the rule is. For a specific association rule mining exercise, minimum support and minimum confidence should be specified to find all the association rules where support and confidence are greater than the provided minimum support and minimum confidence.

The process of mining association rules can be divided into two sub-processes:
(1) Identifying the frequent itemsets: In this sub-process, all itemsets that have support above the specified minimum support are identified. These itemsets are called large itemsets or frequent itemsets.
(2) Identifying the valid/significant rules: in this sup-process, for each frequent itemset, all rules that have confidence more than the specified minimum confidence are identified. For example, if X and Y are frequent itemset, where X, Y ⊂ I, and X ∩ Y = Φ if support (X Y) / support (X) ≥ (specified confidence), then the rule X => Y is derived.

Generally, the efficiency and performance of mining association rules exercise is determined by the first sub-process by which the frequent itemsets are recognized; then the related association rules can be generated in a straightforward manner.

Frequent itemset may contain 2 items, 3 item,…., or k items. The difficult and time consuming part of the mining association rules is finding the frequent k-itemsets; this can be identified if the frequent (k-1)-itemsets are found. Frequent (k-1)itemsets will be used to generate candidate k-itemset. if the candidate k-itemset's support is greater than the specified support, then it will be considered as frequent k-itemset. Frequent (k-2) will be used to identify frequent (K-1) itemset, and so on. Candidate 1-itemset i any item that exists in the transaction database; i.e. any item in the set I where I = {i1 , i2 , . . , im } .

III. APRIOPRI ALGORITHEM

Because "identifying the frequent itemsets" is the most important step in mining the association rules, a variety of algorithms and techniques have been developed to discover the frequent itemsets, see [1,5,6,8,9]. One of the most well known and efficient algorithms in the mining of association rules is the Apriori algorithm. Apriori algorithm, as shown in "Fig. 1", can be outlined as follows:

1-iterations are used to generate the frequent itemsets. In each iteration, the transaction table is scanned one time to generate all frequent itemsets of the same size; ascending order of itemsets is used to generate the frequent itemsets. In the first iteration, the size-1 frequent itemsets(L1)) are generated. then candidate C2 itemsets are generated by using the candidate set generating function Apriori-gen on L1. Subsequently, C k in the kth iteration is generated by using Apriori-gen on L k-1 , where Lk-1 is the set of all frequent (k-1)-itemsets found in iteration k-1. Apriori-gen generates only those k-itemsets whose every (k-1)-itemset subset is in L k-1. The support counts of the candidate itemsets in C k are then generated by searching the transaction table once, and the Lk frequent itemsets are generated from the candidates Ck [8]. For example, let L3 be {A B C}, {A B D}, {A C D}, {A C E}, {B C D}. After executing the joining step, C4 will be {{A B C D},{A C D E}}. After applying the prune step, the following itemsets will be deleted {A C D E} because the itemset {A D E} is not in L3 . Ending with C4 having only {A B C D}.2.Generating rules. For every frequent item set **l**, we output all rules **a** ⇒ (**l-a**), where **a** is a subset of **l**, such that the ratio support (**l**) / support (**a**) is at least minconf

```
L1 = {frequent 1-item sets};
   For (k = 2; Lk-1 ≠ Φ; k++) do begin
   Ck = apriori-gen (Lk-1);          // New
candidates
          For all transactions t ∈ D do begin
 Ct = subset (Ck, t);      // Candidates contained
in t
             For all candidates c ∈ Ct do
                c.count ++;
```

Figure 1: Apriori algorithm (Sarawak, Thomas and Agrawal 1998).

Suppose x is any subset of b, the support of x must be greater or equal the support of b. Therefore, the confidence of x => (l-x) will not be more than the confidence of b => (l-b). Hence, if b did not generate a rule containing all the items in l with b as the antecedent, neither will b. It is known that for a rule b => (l-b) to hold, all rules of the form x => (l-x) must also hold, where x is a nonempty subset of b.

Based on the rule generation algorithm, shown in "Fig.2", for frequent itemset l, all rules with one item in the consequent are first generates. The algorithm then use the consequents of these rules to generate all consequents with two items that can appear in a rule generated from l, etc.

For all frequent k-itemsets l B_{kB}, k ≥ 2, do begin
HB_{1B} = {consequents of rules from l B_{kB} with one item in the consequent}
Call ap-genrules (l B_{kB}, HB_{1B});
End
Procedure ap-genrules (l B_{kB}: frequent k-itemset, H B_{mB}: set of m-item consequents)
If (k > m+1) then begin
H $B_{m+1\,B}$= apriori-gen (H B_{mB});
For all h $B_{m+1\,B}$ ∈ H $B_{m+1,\,B}$do begin
Conf = support (l B_{kB}) / support (l $B_{k\,B}$– h B_{m+1B});
If (conf ≥ minconf) then
Output the rule (l B_{kB} – h B_{m+1B}) ⇒ h B_{m+1}
With confidence = conf and support = support (l B_{kB}); Else
Delete h $B_{m+1\,B}$from H B_{m+1B};
End
Call ap-genrules (l B_{kB}, H B_{m+1B});
End

Figure 2.: Rule generation algorithm (Sarawak, Thomas and Agrawal 1998).

To clarify how Apriori algorithms works, consider the database in Table I, which shows sample transactional database, in this example, minimum support of 50% and minimum confidence of 60% will be used.

TABLE I. SAMPLE TRANSACTION DATABASE

ID	Solid Item
1	1 3 4
2	2 3 5
3	1 2 3 5

Since there are four records in the table, the number of transactions above the minimum support is 2 (4 × 50% = 2). This means any itemset that has 2 or more transactions will be considered a frequent item set.

To generate frequent 1-itemset, the algorithm scan the database and count the support for every item in the transaction databases, the frequent 1-itemset is {1, 2, 3, 5}; item 4 has been deleted because it is exits in only one transaction. Next, frequent k-itemset will be generated by using candidate generation and pruning phases. Candidate 2-itemset are {{1,2}, {1,3}, {1,5}, {2,3}, {2,5}, {3,5}}. The algorithm then calculate the support of each set of candidate 2-itemset and prune the 2-itemset whose support is lower than minimum support to get frequent 2-itemset which is {{1,3}, {2, 3}, {2, 5}, {3, 5}}. The same previous step will used for candidate 3-itemset generation to generate {{2, 3, 5}}; this yields frequent 3-itemset which is {{2, 3, 5}}. Since frequent 4-itemset is empty, frequent k-itemset generation comes to an end. Next, the algorithm derives the association rules for frequent 3-itemset {{2, 3, 5}} where minimum support is 50% and minimum confidence is 60% Table II bellow shows these rules.

TABLE II. SAMPLE TRANSACTION DATABASE

Rule	Support	confidence
2 and 3=>5	50%	100%
2 and =>53	50%	66.7%
3 and =>52	50%	100%
2=>3 and 5	50%	66.7%
3=>2 and 5	50%	66.7&
5=>2 and 3	50%	66.7&

IV. ENHANCED METHOD ON K-WAY METHOD APPROACHE & EXPERIMENTAL RESULTS

Apriori algorithm is implemented using SQL to integrate the algorithm directly with Oracle Database. In this paper, we moved the algorithm forward by enhancing the SQL statement that is used get frequent item sets; to give sound and accurate judgment on our achievement, a performance comparison between our approach and the K-way approach provided by [8] has been conducted by generating frequent itemsets using both approaches on different minimum support values.

The DataSet used is a transaction table named “tt” that has 40028 transactions, 210520 records, 20 different items, 5.25 as an average number of items in transaction. Table III shows the structure of the transactional table .

TABLE III. TRANSACTIONAL TABLE

Column name	Type
Sequence_id	Number
Attribute_name	Char(50)

Generating frequent itemset (F5) has been used in this comparison using candidate itemset C5. “Fig. 3” shows the original SQL statement used by [8] to generate the F5-Itemsets

In this statement, there is a need to join thtransactional table (tt) to itself 5 times. Counting the number of records that would result because of such join (see SQL statement#1 below) ends up with (880985330) records retrieved within 7 minutes. This number in fact is not the correct one that we have to use in counting for frequent items. It is gigantic number because of missing important point that there is no need to join the item in t1 to itself in t2, t3, t4, t5; so instead we used an improved version of SQL statement using “>”

operator to avoid joining the table to itself (see statement#2 below) where the correct number of records is (1839307 records) retrieved in 35 seconds only.

```
Statement#1
select count(*)
from tt t1, tt t2, tt t3, tt t4, tt t5
where
t1.sequence_id=t2.sequence_id and
t2.sequence_id=t3.sequence_id and
t3.sequence_id=t4.sequence_id and
t4.sequence_id=t5.sequence_id;
COUNT (*)
----------
880985330
---------- Execution Time: 7 minutes
```

```
Select i1, i2, i3, i4, i5, count(*) support1
from c5, t t1, t t2, t t3, t t4, t t5
Where
t1.attribute_name= c5.i1 and
t2.attribute_name= c5.i2 and
t3.attribute_name= c5.i3 and
t4.attribute_name= c5.i4 and
t5.attribute_name= c5.i5 and
t1.sequence_id = t2.sequence_id and
t2.sequence_id = t3.sequence_id and
t3.sequence_id = t4.sequence_id and
t4.sequence_id = t5.sequence_id
group by i1, i2, i3, i4, i5
having count(*) > min_support
```

```
statement#2 (enhancement)
select count(*)
from tt t1, tt t2, tt t3, tt t4, tt t5
where (t1.sequence_id=t2.sequence_id
and t2.sequence_id=t3.sequence_id
and t3.sequence_id=t4.sequence_id
and t4.sequence_id=t5.sequence_id)
and (t1.attribute_name>t2.attribute_name
and t2.attribute_name>t3.attribute_name
and t3.attribute_name>t4.attribute_name
and t4.attribute_name>t5.attribute_name);
COUNT(*)
----------
1839307
----------
Execution Time: 35 seconds
```

Figure 3. SQL statement used by K-way method

"Fig. 4" below depicts the execution time (in seconds) for the Apriori algorithm for both cases (K-way method and our enhanced method) using different values of minimum support. As seen in Table IV our proposed method has significant performance enhancement compared to the K-way approach for all specified minimum support values.

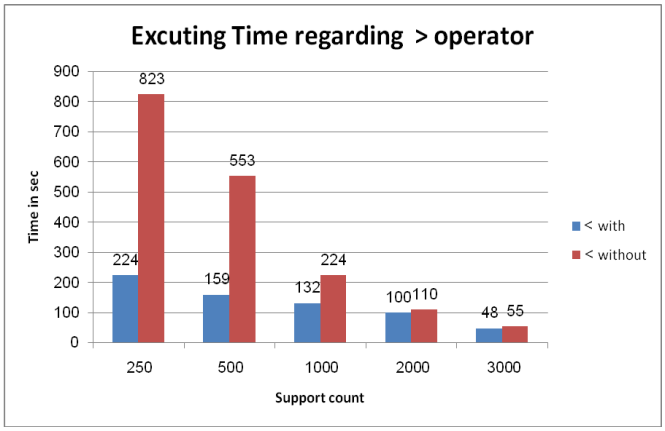

Figure 4. Execution time for K-way method and enhanced method.

TABLE IV. COMPARISON RESULTS

Minimm Support	Execution time K-way Approach (seconds)	Execution time Proposed Approach (seconds)	Time variance (seconds)
250	823	224	599
500	553	159	394
1000	224	132	92
2000	110	100	10
3000	55	48	7

V. CONCLUSION

Where the generation of frequent itemsets is the most important and vital part of association rule mining, we introduce a promoted method for frequent itemset generation through rewriting the SQL statment of frequent itemset generation in a way that can eschew joining the item to itself, where this step is not logic and consumes more time and resources. The process that we have adopted in this paper demonstrated a considerable performance promotion in terms of query execution time, which is less than conventional K-way method.

REFERENCES

[1] R. Agrawal, T. Imielinski, A. Swami. Mining Association Rules between Sets of Items in Large Databases. In Proc. of the ACM SIGMOD Conference on Management of Data, 1993.

[2] R. Agrawal, R. Strikant. Fast Algorithms for Mining Association Rules. In Proc. of the Very Large Database (VLDB) Conference, 1994.

[3] Mirela Danubianu, Stefan Gheorghe Pentiuc, Iolanda Tobolcea. Mining Association Rules Inside a Relational Database – A Case Study. IARIA, 2011.pp14-20

[4] Cyrille Masson, Céline Robardet, Jean-François Boulicaut: Optimizing subset queries: a step towards SQL-based inductive databases for itemsets.in processing of ACM symposium of applied computing SAC 2004: 535-539

[5] Jamil, H.M. Ad hoc association rule mining as SQL3 queries. Proceedings IEEE International Conference on Data Mining, 2001, 609 – 612.

[6] Gang Fang Zu-Kuan Wei Yu-Lu Liu . An algorithm of improved association rules mining. In proceeding of International Conference on Machine Learning and Cybernetics, 2009, 133 - 137

[7] J. Han, Y. Fu, K. Koperski, W. Wang, and O. Zaiane. DMQL: A data mining query language for relational datbases.In Proc. of the 1996 SIGMOD workshop on research issues on data mining and knowledge discovery, Montreal, Canada, May 1996.

[8] Sunita Sarawagi, Shiby Thomas, Rakesh Agrawal, integrating Association rule mining with relational database systems, Proceedings of the 1998 ACM SIGMOD international conference on Management of data, Volume 27 Issue 2.

[9] D. Mirela, G.Stefan, T. PentiucIolanda. Mining Association Rules Inside a Relational Database – A Case Study. The Sixth International Multi-Conference on Computing in the Global Information Technology(ICCGI 2011). June 19-24, 2011 Luxembourg.14-19.

[10] Rao, V.V., R, "Efficient association rule mining using indexing support," Proceedings of the International Conference on Recent Trends in Information Technology (ICRTIT), 3-5 June 2011, Chennai, Tamil Nadu. pp. 683 – 688.

